Refine, improve, and optimize your iOS apps' performance

Pro iOS Apps Performance Optimization

Khang Vo

Apress®

Pro iOS Apps Performance Optimization

Khang Vo

Apress®

Pro iOS Apps Performance Optimization

ISBN-13 (pbk): 978-1-4302-3717-4

ISBN-13 (electronic): 978-1-4302-3718-1

President and Publisher: Paul Manning
Lead Editor: Tom Welsh
Technical Reviewer: Evan Coyne Maloney
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Corbin Collins
Copy Editor: Mary Behr
Compositor: MacPS, LLC
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book’s source code, go to http://www.apress.com/source-code/.

To my girlfriend, Ngan Huynh,
for the love and great encouragement and support.

Contents at a Glance

Contents

About the Author

Khang Vo is a software engineer and entrepreneur who loves working on the latest technologies and products. He has been developing on the iOS platform since 2009. He loves sharing and discussing different aspects of technology and business that help to create new value for consumers. Making and selling different applications in the Apple App Store and Android Market have been his main business. He is a Master's student at Carnegie Mellon University.

About the Technical Reviewer

Evan Coyne Maloney taught himself how to program after inheriting an Apple IIe computer. As a young teen in the mid-1980s, he published an operating system for the Apple II line called FoscilDOS that was highlighted by both *Byte* and *A+* magazines. Evan began writing Internet software in 1994, creating the KeepTalking chat system, the first purely browser-based live-updating chat system. In 1996, Evan wrote the web-based political campaign simulation game DarkHorse for MSNBC.com. During the presidential campaign that year, the game logged many millions of hours of play and was even used in political science classes at various high schools and colleges. Since 2001, Evan has put his development efforts towards mobile content delivery and commerce. He conceived of and built the first several versions of the award-winning News Pro line of iPhone and iPad applications from the Reuters news agency. Evan joined online retailer Gilt Groupe in 2010, where he is now the principal engineer for the company's critically acclaimed and highly ranked iOS applications.

Acknowledgments

I would like to thank Steve Anglin, who approached me with the idea of creating this book and guided me through the initial process. I also want to thank Evan Maloney for providing many helpful suggestions over the technical part of the book. I learned many things from Evan while writing this book.

Thanks also to Tom Welsh, who helped me make the writing clear and easy for readers to understand. He has made lots of great suggestions to guide the book into its final form. I also want to thank Corbin Collins for his quick and helpful instruction when I asked questions.

I also thank developers and people who shared and helped me with technical difficulties in writing the book. It helped me to figure out what developers lack and how to help them get the necessary knowledge and skills.

Preface

The book is meant to help you to sharpen your iOS development skills in a specific area: performance optimization. The book is intended for people who already have basic skills in iOS development and want to make the best application for users.

Insipired by the art of application performance, I spend time practicing, learning, and sharing a lot about performance optimization in different platforms such as the web and smartphones. I love discussing this topic with people. While spending lots of time in forums and iOS communities like Stack Overflow and the Apple Developer Forum, I soon recognized that the majority of iOS developers have the same questions on how to improve the performance of their applications. I thought it would be useful to put all common issues together in a well-written and well-structured book so people can easily get the whole picture of the iOS performance optimization problem. That motivated me to write this book, and I tried my best to cover the most common problems and mistakes met by developers.

Moreover, I observed and record in my own notes many similar problems between iOS, Android, and Windows phones. The final chapter is written based on these notes, and I think this chapter will be really useful for anybody who wants to work in these three platforms or shift from one platform to another.

When approaching a performance bottleneck, it is good to see it in different ways and strike a balance between the performance of the application and the difficulty of implementing the solution. There are subtle problems that cause people to make mistakes unless they know about the solutions beforehand. My hope is that this text will help you to avoid those mistakes, spend your time improving your application, and create a better experience for your users.

Chapter 1

Introduction to iOS Performance Optimization

This chapter will introduce general information about the book, including the following:

- Who this book will best serve
- The topics this book will cover
- The general structure and style of the book

A New Era of Smartphone

There are currently hundreds of thousands of iOS applications on the market and hundreds of millions of iOS users, making this a big market for any company or developer to explore. This market has been growing for many years and will keep growing in the next few years, as will the need for interesting and powerful applications. If you have a good idea for a new app, you need to make sure that the idea is implemented well; this includes creating a good user experience. Because of the unique technical limits of the Smartphone environment, good performance is a must for your application. People want an app that responds quickly to their interactions, one that can compute data and visualize it immediately.

Why Performance Matters

Performance is not just about algorithms, data structure, and memory. It's about making people feel that the application responds to any interaction as fast as possible. Therefore, performance optimization in your iPhone application is important. Users have to feel that they are interacting with real agents that receive their command and execute it almost immediately. What if you tap a button and two long seconds later, you see the effect. Are you happy with this performance? If you're not happy, your users are probably even more frustrated.

Of course, you can shift much of your storage and processing into the cloud where there are thousands of servers that can compute and return the result quickly. However, it's not enough to just put all your data and every computation into the cloud. Network data transfers are tricky and your users will still probably need to wait for couple of seconds before their data arrives.

Whether you are a game developer or a general application developer, you are likely to experience difficulties in improving the performance of your applications.

Who Should Use This Book?

This book is written mainly for beginner and intermediate iPhone developers who already know basic iPhone programming. If you're a performance lover and want to create an application on this new platform that is responsive and market-ready as well as innovative, this book is for you. Even advanced iOS developers can benefit from this book.

If you intend to go deeply into the Smartphone application programming world, this book teaches you enough so that you can apply what you know with iOS to Android and Windows Phone environment.

My Teaching Style

I believe that the learn-by-doing principle is the best way for a programmer to develop skills. This book is based on that idea. I discuss general and deep practices that stem directly from around two years of iPhone development experience and many years of Java development experience and training. The problems that I put before you will help you to avoid or fix many of your performance mistakes in iPhone development. I have chosen these problems based on experience and research into the issues popular on forums and social networks (such as Stack Overflow). I've identified common pitfalls and provide the information you need to avoid these errors.

The book is a combination of three things: basic concepts, story illustrations, and sample source code. Instead of just supplying an ad hoc tool for your specific problem, I hope to provide you with strong skills to use in your daily iPhone programming life. I employ different approaches to communicating concepts: sometimes an image is worth a thousand of words, some concepts are best explained by sample code, and some require those thousand words.

One of the best ways to start learning about performance is to develop a cool application that you love. This practical experience will teach you more than some non-realistic and very forgettable examples.

You don't need to know a lot about Cocoa Touch Framework because I explain the basic syntax and classes that you'll need to improve your application's performance. Each chapter consists of a separate topic, some of which may already be familiar to you.

You can also use this book as a general reference; whenever you have a specific problem, you can look it up and read about the solution.

Every chapter follows a simple format: a short overview about what that chapter delivers followed by the main sections and subsections. Each chapter concludes with a summary that helps consolidate your knowledge and reminds you of the important lessons, followed by some basic and realistic exercises so that you can have fun practicing what you just learned.

What Do You Need?

As an iOS developer, you need a Mac OS with Xcode installed. There is a free Xcode version from the iOS Developer Account, or you can download it directly from Apple Mac AppStore. You also need a copy of this book plus all of the sample code, which you can download from the Apress website. The sample projects were well tested on Xcode 4.2, with ARC turned on, so you can run my sample projects in that environment without any concern.

You can and should run every example to understand more about the illustrated concepts. There are some short blocks of code that aren't associated with any project; you should run that code, too.

How to Use This Book

Although the chapters are not closely related, reading the book from beginning to end will ensure that you have a solid knowledge of iPhone performance, optimization skills, and techniques. There may be some dependencies and references between chapters. The later chapters were written with the assumption that you have read or know about some previous chapters.

I also recommend reading each chapter from beginning to end. Each chapter opens with a quick conceptual introduction to the topic; then theory and practical iPhone samples are combined to help you to understand the topic thoroughly.

You should read the summary section carefully because it reminds you of the key knowledge that you should retain. I also recommend finishing all of the exercises as these will help cement your new knowledge.

An Overview of the Book

This book contains a good mix of basic concepts plus practical knowledge, techniques, and tips that will help you to be successful in the competitive iOS development world. The book's nine chapters cover nine different approaches to solving performance problems in iOS development.

- *Chapter 2:* The introduction to a range of tools and instruments so that you know how and when to use them. Many developers don't use these tools correctly because they simply don't know that they exist.
- *Chapter 3:* As an iOS developer, you will definitely use TableView in almost all of your projects, from trivial ones to complex ones, to display a list of data or options. The problem with the architecture of `UITableView` is that when you start customizing it, the scrolling performance suffers. You will definitely have this issue, even if in a subtle way. This chapter gives you a list of tools and techniques to improve your TableView scrolling.
- *Chapter 4:* You may believe that most performance issues can be solved using cloud computing and by simply adding more servers to your system. Even if that's true, network data transfer will always be an issue. Data transfer will remain a bottleneck for years. You should understand how to cache data locally and in memory with a limited environment like iOS.
- *Chapter 5:* Data structures and algorithms in the iOS development environment are similar and different than in other environments. You have a high level of support from the framework with many basic data structures like arrays, sets, and dictionaries. For some tasks, you can simply put it to the cloud, but for other tasks, especially gathering and processing data to make a good visualization, you still need to depend on the iOS environment.
- *Chapter 6:* Improving the performance of the application also means making the application respond to users' interaction faster. This means not blocking the main UI thread. Multithreading can help—not just to improve the user responsiveness but also to improve the general performance of your application. Multithreading is a difficult topic for any platform, and you will learn it here through a range of illustrations, examples, and clear explanation.
- *Chapter 7:* With the release of a new tool to make memory management automatic, developers now can take advantage of it to avoid common memory problems like memory leaks and crash. This chapter focuses on how you can best use your memory, and when you should load data in and unload data out of your memory. It also covers the new Automatic Reference Counting (ARC) mechanism of the new SDK to make sure you can understand and use it correctly.
- *Chapter 8:* With iOS 4 and above, all applications can take advantage of multitasking to improve the user experience. In fact, it's not actually multitasking but rather a fast app-switching mechanism (applications can't run in background) with some special background processing. This chapter will help you understand what features the iOS will support and what tasks you can process and run in background.

- *Chapter 9:* In many iPhone applications, you don't need to use any C/C++ code to implement features. However, when you actually need it, especially for library integration, you will be in serious trouble. You may not need to write your whole application in C/C++ but you do need to understand how these languages work for any necessary troubleshooting.
- *Chapter 10:* By now you should have a complete picture of all the different aspects of iPhone performance. You will definitely consider porting your application to Android and Windows Phone soon, so in this final chapter, I give you the whole picture on similarities between performance problems in iOS, Android, and Windows Phone. This will help to smooth your learning experience for new platforms.

Source Code

You should download the sample source code from the book's page on the Apress web site (`www.apress.com`) and try it on your own.

Contact the Author

If you have any questions, please email me at `vodkhang@gmail.com` or visit my web site at `http://vodkhang.com`. I shall be happy to have a chat about iPhone performance problems.

Chapter 2

Benchmark Your Apps with Tools: Simulators and Real Device Test

In this chapter, you will learn about the following:

- The differences between a simulator and real device test environment.
- How memory management affects the performance of an app.
- Tools and techniques to benchmark your app's performance including the following:
  - Basic tools to measure the memory and performance.
  - Complicated tools to measure different aspects of memory management such as memory leaks and bad access.
  - Complicated tools to measure different aspects of performance in computer processing such as battery, file loading, and display information.
  - How to divide your program into smaller parts to easily identify the location of the performance bottleneck.

To improve performance, you need to carefully run benchmark tests to see where the problems lie. To carry out a useful benchmark test, you have to understand the different reasons that a program or a segment of code might run slowly.

Right at the outset, you should be aware of two fundamental choices: simulator versus real device environment, and the trade-offs between memory optimization and performance optimization.

First, you need to know the difference between the simulator and device environment.

Simulator and Device

The main problem with the performance of iPhone applications is that they are running in a restricted, slow-processing environment. The iPhone development environment simulator runs much faster than the real environment; in fact, the simulator's environment can be as fast as the machine running it.

As a result, you can get a big and unpleasant surprise when the program runs really fast in the simulator environment but runs much slower in the real environment. I have observed many people blaming slow application performance on the phone's network. This is certainly true in some cases. However, in many cases the app's performance can drop down a lot because of the code implementation itself, not because of a network problem. Therefore, careful testing and benchmarking your app against basic tools and standard environments will make you more confident about your app's performance and the user experience.

To demonstrate the significant differences between the simulator and real device, I tested a program in the iPhone simulator environment and the real iPhone environment. The results are surprising.

- It takes 0.5 seconds to finish the main calculation in the iPhone simulator.
- It takes 7 seconds to finish the same calculation on the iPhone device.

The program was simple: I did a simple test with two arrays, each with 1000 elements. Then, the code loops over both arrays to find the same number and print "hello." In the real world, you may not need to process 1000 items in an array or you may not choose to loop over arrays to find same number. However, this is not the point. I picked these actions to demonstrate that real iPhone environment is much slower than in the iPhone simulator.

This brings me to a point that I will mention many times in this book: you always need to test the app on both simulator and the real device. Well, why not just test on the device? Because simulator has the following significant benefits:

- It is faster to run the test in the simulator, which means less delay time for developers.
- It is good enough to test for memory leaks and memory allocation problems.

Memory and Performance

Memory and performance are different. Memory usually means the RAM storage, and it refers to how much storage you use and how much you have left. Performance is about how fast your app runs a specific feature.

Memory can have a significant effect on performance. When the device has more RAM and more storage space, you can preload and cache more data on it. RAM is fast access storage compared to file storage and the network. By preloading and caching

more data on RAM, you can significantly speed up your program in many cases. For example, if your app is a game that needs to load many images, more memory is important to because you can preload the images and display them when necessary. Loading from RAM is 10 times faster than loading from the file system.

However, better use of memory does not always mean better performance. Some apps don't need to use much memory; therefore, you can optimize the memory only so far and the performance will not go up anymore. The inverse is not much better: an app can use up all the memory in order to achieve good performance, but then the app runs out of memory.

Therefore, you should always carefully benchmark both the memory and runtime performance to make sure that you strike a good balance between memory usage and runtime performance.

Tools

The tools fall into the following three main categories:

- Basic tools, without XCode instruments.
- Memory tools, which verify the correctness and measure the efficiency of your memory usage.
- Performance tools, which measure how fast each part of your program runs and pinpoint any bottlenecks.

Basic Tools

In this part, I discuss about logging as a basic tool to measure the running time between blocks of code.

Logging the Running Time

One of the most basic tools is logging the time difference between the start and end of a block of code. Usually, logging is implemented with NSLog. With this basic tool, developers can measure every line of code or block of code to see how fast that block of code runs.

For example, running this block of code

```
NSDate *date1 = [NSDate date];
for (int i = 0; i < 1000; i++) {
  // Do calculation here
}
NSDate *date2 = [NSDate date];
NSLog (@"time: %f", [date2 timeIntervalSinceDate:date1]);
```

returns this result

```
time: 0.0123 (measured in seconds)
```

Advantages:

- A straightforward and easy way to measure the performance.
- You can measure the performance of lines of code or blocks of code.

Disadvantages:

- You can't measure the UI performance (i.e. the rendering time of the UI thread).
- You can over-optimize (spend too much time on a very specific block of code just to optimize it a little bit).
- Running the application in simulator is usually fast, and at this fast level, NSLog can't help you distinguish between a difference in runtime performance. Otherwise, although NSLog is slow in the device, it can help you to detect the differences in runtime performance.

Usage:

- When you need an immediate tool to measure without much planning.
- When you need a tool that can return a result quickly.
- When you need to isolate a small block of code to verify a performance assumption.

Memory Tools

With memory problems, you have only one main concern: high memory usage. There are minor concerns with legacy code: memory leak and memory garbage. For the new projects, you should go straight with the new Automatic Reference Counting (ARC in short) support. For some old projects, you can try to convert them using the convert tool of Xcode.

However, not every project can be converted, there are many issues and memory management policy that prevents you from conversion. Trying to comply with the new management policy may cause you more troubles. So, I discuss mainly with you about the memory tools for object allocation and briefly about tools for memory leaks and memory garbage.

> **NOTE:** All the memory tools that I introduce here (and I introduce all Apple's tools for memory) can be run with simulator. The good thing about the simulator is that it runs really fast and installs apps quickly. However, be careful! I strongly recommend that you also test your apps on the device because the simulator and device are not always the same. They are built differently and have different architectures.

Memory Allocation

Memory Allocation helps you to understand how much object allocations you use. This may mean that you allocate and keep in memory so many objects. These objects are not released yet because it is still in use.

Allocation

Choose Product ➤ Profile and then choose Allocations in the open window (as shown in Figure 2–1)

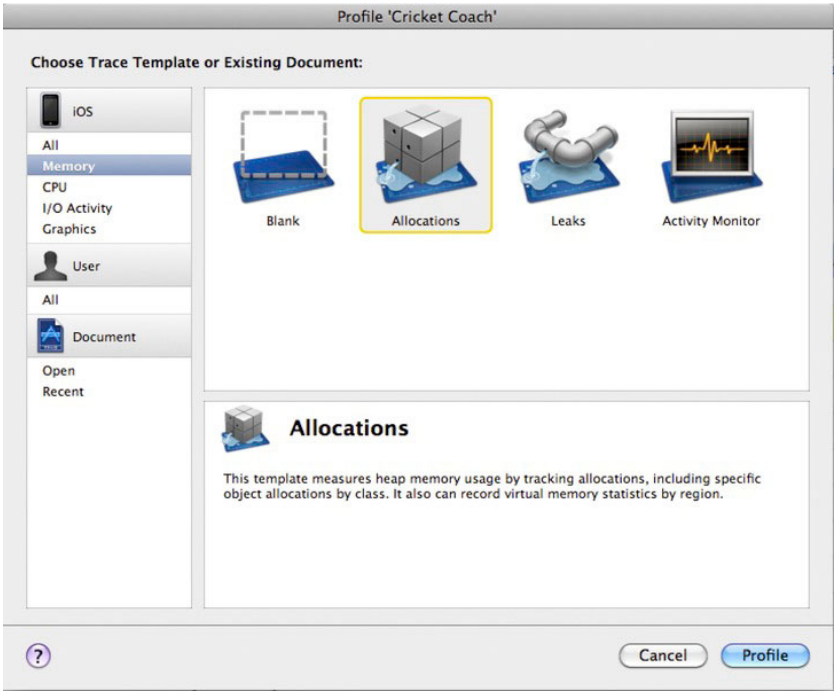

Figure 2–1. *Choose Allocation in Profile Window*

After choosing Allocations instrument, you will be shown with a main Allocation panel, which gives you all the necessary information, as you can see in Figure 2–2.

The Allocations panel (Figure 2–2) shows you “created and still living” jobs so that you can see what objects are still in memory and what objects consume the most memory. You should use this tool if you start receiving many warnings from the iOS environment like “Received memory warning. Level =1”.

The details will show you at which time which lines of code and which class is responsible for creating and handling the objects. With this information, you can easily figure out how to deal with memory. This is a good tool for tracking caching algorithms and methods (see Chapter 4 for more details).

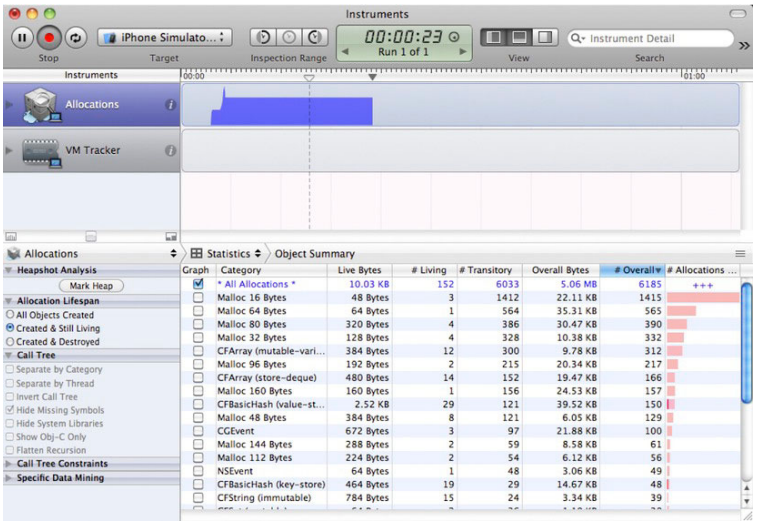

Figure 2–2. The main allocations panel

Figure 2–3 and 2–4 shows you more details about what objects are living and consuming the most memory for your application. In Figure 2–3, you see the list of details about objects are created and lived inside your application.

Graph	Category	Live Bytes	# Living	# Transitory	Overall Bytes	# Overall	# Allocations (Net / Overall)
☑	* All Allocations *	27.77 KB	565	55386	17.06 MB	55951	
☐	CFRunLoopTimer	11.72 KB	125	250	35.16 KB	375	
☐	CFBasicHash (value-st...	4.50 KB	135	1146	109.23 KB	1281	
☐	CFSet (mutable)	3.91 KB	125	266	12.22 KB	391	
☐	CFBasicHash (count-s...	2.36 KB	8	57	16.86 KB	65	
☐	__NSCFDate	1.94 KB	124	247	5.80 KB	371	
☐	CGEvent	672 Bytes	3	291	64.31 KB	294	
☐	Malloc 144 Bytes	576 Bytes	4	924	130.50 KB	928	
☐	Malloc 48 Bytes	480 Bytes	10	1353	63.89 KB	1363	
☐	Malloc 80 Bytes	320 Bytes	4	4787	374.30 KB	4791	

Figure 2–3. The allocation results

In Figure 2–4, you can see which methods are calling to create these objects.

Category	Timestamp	Live	Size	Responsible Library	Responsible Caller
Malloc 80 Bytes	00:22.186.397	•	80	CoreGraphics	CGClipStackCreateMutable
Malloc 80 Bytes	00:25.680.592	•	80	CoreGraphics	CGClipStackCreateMutable
Malloc 80 Bytes	00:26.680.849	•	80	AppKit	-[NSViewHierarchyLock lock...
Malloc 80 Bytes	00:26.680.980	•	80	CoreGraphics	CGClipStackCreateMutable

Figure 2–4. The allocation details

Advantages:

- It is accurate and provides many details on the time and situation in which the application consumes the most memory.
- It can also give you a good overview of the object's lifecycle over the application lifetime.

Disadvantages:

- The results depend on how developers run the app. It requires a good test suite preparation to cover as many cases as possible.
- It can take time and effort to create a good test case that helps developers to figure out the place and time the application consume the most memory.
- You need to test on the real device so that you can receive memory warning message. The simulator will almost never give the memory warning message. The problem with using the simulator is that your computer will have 2–4GB of RAM and your device probably has much less.

Usages:

- If you test your app and receive a memory warning, this is the one of the first tools you should reach for.

Legacy Code

At this release, the tool to automatically convert from a manual memory management project to new ARC project may fail. The tool may ask you to fix lots of places in your current code to make sure the project can be converted into an ARC project. It may be your open source library fails to convert into a new ARC style and you would not want to touch it. So, I think it is good for you to understand some background about manual memory management.

Memory Leaks

Memory leaks happen when you create a new object in memory and you don't release it properly. That object will stay in memory for the whole application life. The result is your application doesn't have enough memory to run fast, or even worse, the iOS will force your application to close.

Static Analyzer

This is a simple and straightforward tool to measure the memory leaks. As shown in the Figures 2–5 and 2–6, the tool will show you which line or block of code may **possibly** be causing the memory leak.

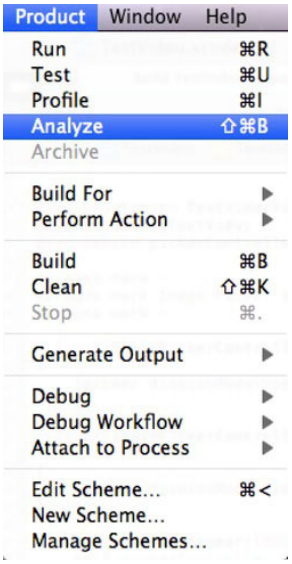

Figure 2–5. *Choose Product ➤ Analyze*

As shown in Figure 2–5, you need to choose Product ➤ Analyze or Command + Shift + B

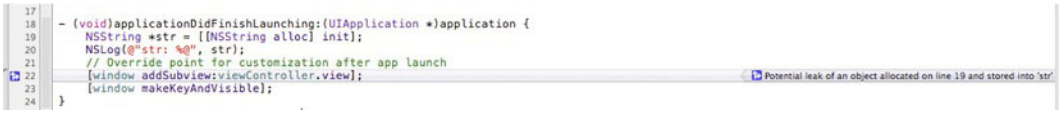

Figure 2–6. *Static Analyzer reports a potential leak of an object allocation on line 19 and stored in str.*

As you can see in Figure 2–6, the `str` object in line 19 is never released; in this case, Static Analyzer provides a correct warning.

Advantages:

- It gives you a quick and general look at possible places where memory leaks can happen.
- It has a really fast process: it only builds and looks at the source code. Static Analyzer doesn't need to run the program.
- This tool requires no effort from the developer; you just click on Build and Analyze.

Disadvantages:

- Sometimes it's not accurate. It can give an incorrect warning or doesn't indicate places where there is a memory leak.

Usage:

- Developers should use this tool first to measure the memory leak because it's fast and requires almost no effort.

Leaks Instrument

This is a better instrument that measures the memory leak in runtime (when the app is running). This makes sure that the object is really leaked out; if an object is leaked out, it will have to be reported to the user. You keep trying different features of the app, and the Leaks Instrument will report memory leak places.

You will need to look for places where the Leaks horizontal bar shows a vertical column. The height of the column will show how much memory the app has leaked at that time (see Figure 2–7).

Figure 2–7. Shows how many leaks you have had from running the code

Then, when you go inside the details of the leaks, you may see a list of leaks happening in your code. By sorting by responsible library and looking for your app name (in this case, `LeaksViewController`), you will see two leaked objects. A quick look tells you that you leaked two images inside the class `RootViewController`.

Leaked Object	#	Address	Size	Responsible Library	Responsible Frame
UIImage		0x8026020	16 Bytes	LeaksViewController	-[RootViewController
UIImage		0x4b2a250	16 Bytes	LeaksViewController	-[RootViewController
Malloc 9.00 KB		0x5026a00	9.00 KB	ImageIO	initImageJPEG

Figure 2–8. A list of leaks inside your program

As shown in Figure 2–8, next to the address is a small arrow; by clicking on it, the Leaks Instrument will guide you to the correct place in the app that caused that leak (see Figure 2–9).

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    NSString *avatarFile = [NSString stringWithFormat:@"a0"];
    NSString *avatarName = [[NSBundle mainBundle] pathForResource:avatarFile ofType:@"jpeg"];
    UIImage *image1 = [[UIImage alloc] initWithContentsOfFile:avatarName];
    NSLog(@"image: %@", image1);
    image1 = [[UIImage alloc] initWithContentsOfFile:avatarName];
    NSLog(@"image: %@", image1);
```

Figure 2–9. Lines of code that created the leaks

At this point, you can observe the line of code that created the memory leak. Usually, Leaks Instrument will give you exact details about where the memory leak happened so that you can easily fix it.

Advantages:

- Leaks Instrument is very accurate and detailed.

Disadvantages:

- The results depend on how developers run the app. It requires a good test suite preparation to cover as many cases as possible.
- It can be slow because developers need to run it a few times to see how the app performs in many different cases.

Usages:

- This tool should be used after the Static Analyzer is used. It will cover all other small and niche cases that the Static Analyzer missed.

I recommend that you run Static Analyzer first. If you still have some concerns over memory usage or receive memory warning from the iOS runtime environment, you should use Leaks Instrument.

Memory Garbage

At the first look, memory garbage may not seem to be related to performance issues. However, having your application crash is even worse than slow performance as it stops performance cold and kills the whole user experience that you want to create. Therefore, you should know how make the best use of memory.

Zombie

You choose Product ➤ Profile ➤ Allocations.

You will be shown a running instrument. The problem is that this instrument does not measure anything or help you with the Zombie issue, so you need to stop it. Then, you need to configure the Allocations to work with the Zombie. In other words, when a crash happens, the Instrument will report where the crash happens.

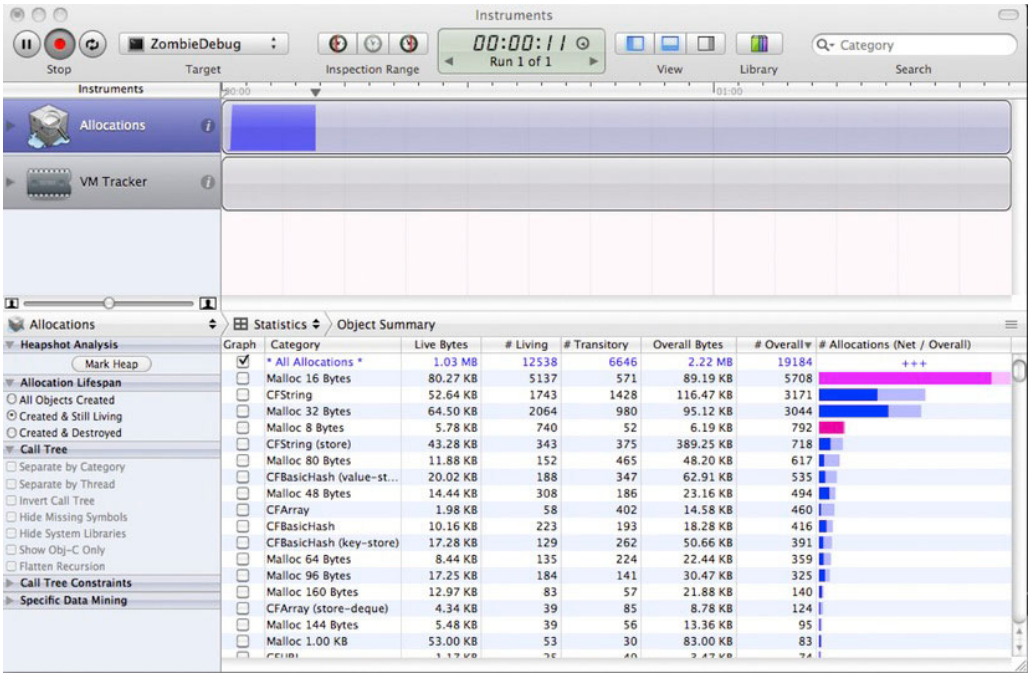

Figure 2–10. The screen for Allocations Instrument